Enhancing Productivity and Security: A Comprehensive Strategy for Local Large Language Model (LLM) Deployment

Executive Summary

This white paper explores the feasibility and benefits of deploying local LLMs within our organization, emphasizing security and compliance. Leveraging the power of open-source software, we can run AI/LLMs locally on existing company hardware such as MacBook Pro M2 and M3 laptops without compromising data security or breaching organizational policies. This approach presents a case for allowing local LLMs in specific contexts while addressing concerns related to data protection, access control, model updates, monitoring, and auditability. By adopting this secure and compliant strategy, we can enhance AI-driven productivity, improve security measures, and comply with existing organizational policies.

1. Enhanced Security through Local LLMs

- **Data Protection:** Local LLMs ensure that sensitive data never leaves the user's machine, mitigating risks associated with data leaks and third-party training of AI models. This aligns with our Personal Data Protection Standard, which mandates that personal data must be protected from unauthorized access and processing.
- **Access Control:** By utilizing locally run LLMs, we can maintain control over data security and compliance procedures without outsourcing this responsibility to third parties. This approach supports our Information Governance & Classification Standard that requires restricting access based on the principle of "least privilege".

2. Compliance with Organizational Policies

- **Global AI Standard (P-RS-GL-GET-008) Compliance:** Local LLM deployment meets the standards set by our organization's Global AI Standard policy, ensuring secure and compliant data processing. This includes adhering to the requirements for responsible and ethical AI use, addressing potential negative impacts, and ensuring data used is classified appropriately.
- **Bring Your Own Hardware (BYOH) Policy Compliance:** By leveraging existing company hardware and our BYOH policy, we can achieve the benefits of AI-driven productivity while ensuring the safety and integrity of company data. The primary requirements of the BYOH policy include:
    - **Device Security:** Ensuring that any personal devices used for company work are secured according to company standards.
    - **Data Protection:** Personal devices must have encryption enabled on all storage devices to protect any company data stored on them.

- **Software Compliance:** Only approved software should be used on personal devices to maintain security and compliance.
- **Access Control:** Personal devices should have access controls, such as passwords and biometric authentication, to prevent unauthorized access.
- **Monitoring and Maintenance:** Personal devices used for work should be regularly monitored and maintained to ensure they meet security and performance standards.
- **Antivirus and Security Tooling:** Crowdstrike Falcon sensor MUST be installed and running on any equipment used for work.

3. Implications and Benefits

- **Enhanced Productivity:** Local LLMs accelerate AI-driven productivity, enabling faster development cycles, improved accuracy, and enhanced decision-making capabilities.
- **Innovation and Growth:** By deploying local LLMs securely and compliantly, we can harness the power of AI to drive innovation and growth within our organization.
- **No Additional Cost:** Utilizing existing company hardware for local LLMs incurs no additional costs beyond the initial investment in equipment.

4. Emphasizing the Importance of Proper Use

While Large Language Models (LLMs) have inherent limitations, their proper integration into the workplace can only be achieved through practical application. Adherence to our established guidelines, as outlined in the Global AI Standard and the NIST AI Risk Management Framework (AI RMF 1.0), will ensure the responsible, ethical, and safe deployment of AI systems.

Best Practices When Using LLMs:

1. **Test All Code Thoroughly**
   Explanation: Rigorous testing of any code generated by LLMs is essential. While LLMs can produce functional code snippets, they may not always meet the specific requirements or nuances of your project. Thorough testing ensures the code performs as expected, is free from bugs, and integrates seamlessly with your existing systems.

   - ***Action Steps:***
     - Implement unit tests, integration tests, and system tests.
     - Regularly review and update test cases to cover new functionalities or changes in the code.

2. **Ensure the Logic of the Code is Human-Generated**
   Explanation: The core logic and structure of your code should be designed and conceptualized by humans. LLMs are powerful tools for generating syntax but lack the contextual understanding and creative problem-solving abilities of humans. This ensures solutions are tailored to real-world problems and are robust.

- ***Action Steps:***
  - Use LLMs to generate boilerplate code or repetitive syntax.
  - Focus human effort on defining algorithms, data structures, and overall system architecture.
  - Regularly review and refine the logic to align with project goals and requirements.

3. **Review All Output Before Dissemination**
   Explanation: Always review the output produced by LLMs before sharing it with others or integrating it into your systems. This is crucial to catch any errors, inaccuracies, or unintended consequences that the model might produce.

- ***Action Steps:***
  - Implement a peer review process as one would for any code created on a team.
  - Use automated tools to check for common issues such as security vulnerabilities or performance bottlenecks.

4. **Confirm All References Listed in Any Output**
   Explanation: LLMs can generate references to external sources, but it's essential to verify these references for accuracy and credibility. Misinformation can undermine the integrity of your work and lead to serious consequences.

- ***Action Steps:***
  - Manually check each reference for validity and relevance.
  - Use reputable databases and citation tools to cross-verify references.

5. **Use Docker or Full Virtualization for Improved Security and Data Segregation**
   Explanation: Utilizing Docker or full virtualization provides a secure and isolated environment for running LLMs. This helps protect sensitive data, ensures consistent performance, and prevents potential security breaches.

- ***Action Steps:***
  - Set up Docker containers or virtual machines for different components of your application.
  - Regularly update and patch these environments to mitigate security risks.
  - Use network segmentation and access controls to further enhance security.

Human Solutions for Human Problems Using LLMs

Human problems require human solutions. LLMs are tools designed to assist us. While they can help generate code, content, or ideas, the underlying logic, strategy, and problem-solving should come from humans. Human intuition, creativity, and contextual understanding cannot be replicated by LLMs.

Personal Responsibility and Ownership

Leveraging LLMs does not absolve us of responsibility for the output we generate. Regardless of whether content is created by an LLM or a human, it is ultimately our responsibility to ensure its accuracy, reliability, and integrity. This means taking ownership of the entire process and being accountable for the final product.

- ***Action Steps:***
    - Emphasize the role of humans in the initial planning and final review stages.
    - Foster a culture of accountability where individuals are responsible for the outputs they oversee, whether generated by LLMs or manually.
    - Continuously educate and train team members on the ethical and responsible use of LLMs.
    - Leverage the existing culture of responsibility and ownership to promote the safe and ethical use of AI/LLMs in the workplace.

Example Implementations

Global AI Standard
Before deploying an LLM, conduct a comprehensive risk assessment to identify potential negative impacts, such as bias or incorrect outputs. Implement a review process where the output is evaluated by both AI specialists and domain experts to ensure accuracy and relevance. Document these processes to maintain transparency and accountability. Leading LLM models setting industry standards include LLama3 (Meta), Codestral (Mystaral AI), WizardCoder (Microsoft), and Grok (Tesla).

NIST AI Risk Management Framework
Apply the NIST AI RMF by integrating risk management practices into the AI development lifecycle. This includes regular audits, monitoring model performance, and updating the model based on new data and feedback. Ensure all data used for training the LLM is anonymized and compliant with data protection standards, and continuously monitor for any deviations or unexpected behaviors during deployment.

5. The Difference Between Training and Fine-Tuning

Training an LLM involves creating a model from scratch using large datasets, typically requiring significant computational resources and time. This process helps the model learn the complexities of language, grammar, and context. Fine-tuning, on the other hand, involves taking a pre-trained model and further training it on a smaller, specific dataset to adapt it to a particular task or domain. While training builds the foundational capabilities of the model, fine-tuning tailors it to specialized needs.

6. Technical Approach and Hardware Requirements

I built (using equipment I had on hand and what I could purchase) two proof-of-concept machines to test the physical requirements for running large language models (LLMs). The first machine was designed to handle training, fine-tuning, and inference of smaller to medium-sized LLMs. The second workstation was intended for inferencing small to medium-sized LLMs with limited training capabilities. Both machines used Ubuntu 22.04 as the OS.

Training Machine:

- ASRock TRX50 WS motherboard
- AMD Threadripper 7960x processor (24 cores)
- 256GB of DDR5 EEC RDIMM memory
- Nvidia RTX A6000 GPU with 48GB VRAM (Ampere architecture)
- Nvidia RTX 4090 GPU with 24GB VRAM (Ada architecture)
- Crucial T700 4TB Gen5 NVMe M.2 SSD (LLM Datastore)

Commented [TB1]: Oh how naive I was. my first tests were with relatively small datasets of less than 10GB in size. However the more useful datasets can get quite large and exceed the capabilities of my current system.
we should focus on inference and any talk of specific training should be directed towards the FAIR team.

Inference Workstation:

- MSI MPG B550 AM4 motherboard
- AMD Ryzen 7 5900X processor (12 cores)
- 32 GB of DDR4 RAM
- Nvidia RTX 1080 ti GPU with 12GB VRAM (Pascal architecture)
- Crucial T500 1TB Gen4 NVMe M.2 (LLM Datastore)

While the smaller inference workstation could run small 7 and 14 billion parameter models, performance was severely limited by a lack of VRAM, CUDA, and Tensor cores. This should be considered the absolute bare minimum for inferencing only. Thus, what I have listed below is what I would consider a minimum requirement for production inference work.

Minimum Requirements:

- Smaller LLM models require a Nvidia GPU from the 20 or 30 series with at least 16 GB of RAM (e.g., RTX 2080 Ti, Titan RTX, RTX 3080 Ti etc.)
- A multi-core processor with a minimum clock speed of 2.5 GHz. minimum 8 cores.
- 32 GB of DDR4 or DDR5 memory.
- High-speed storage such as NVME SSDs with a minimum capacity of 1 TB

Note that these are minimum requirements, and larger LLMs may require significantly more powerful hardware resources. As technology continues to rapidly improve, we should expect to see performance improve and hardware requirements decrease over time.

6. Model Selection and Recommendations

Commented [TB2]: This partially addresses the weak point I listed earlier, but I don't think it's enough. I still believe we should have something more robust than third party performance benchmarks.

Commented [TB3R2]: Maybe we can address this in a more agentic way. What if we had a system of internal LLMs that could be distributed? These LLMs would be the ones who are trained on non-sensitive issues. On the Racker system, LLM could act as an agent such that when it needed access to sensitive information, it could reach out to the company LLM/AI running in one of our secured data centers. Security protocols between these LLMs could then be set up to verify authorization and access.
So, in this scenario, the only LLM.AI that is trained on internal company data is the one running in our secure datacenter. The LLM/AIs running on Racker workstations are not trained on that data but on how to query for that data from the internal AI/LLM.

Commented [TB4R2]: the benefit of doing this is that you now spread your computer cost across a wide range of devices instead of consolidating them into a single silo.

To effectively deploy local LLMs, careful model selection is crucial. Below are recommendations for different use cases:

- **Codestral and CodeLama:** Ideal for code generation and assistance.
- **LLama 3, Falcon 2, and Vicuna-13B:** Best for technical documentation.
- **Mixtral-8x7B and Mistral LLM:** Excellent for math and reasoning tasks.

Codestral

Codestral, developed by Mistral AI, supports over 80 programming languages, including Python, Java, Swift, and Fortran. Its fill-in-the-middle mechanism completes coding functions, writes tests, and fills in partial code, significantly reducing coding time and effort. With a 22-billion-parameter model, Codestral outperforms previous models in latency and capability. It integrates seamlessly with popular development tools like VS Code and JetBrains, making it versatile for developers.

Key Features:

- Supports over 80 programming languages.
- Fill-in-the-middle mechanism for code completion.
- 22-billion-parameter model for high performance.
- Integration with popular development tools.

Code Llama

Code Llama, developed by Meta, is a specialized version of the Llama 2 model for coding. It comes in three sizes (7B, 13B, and 34B parameters) and variants (base, Python fine-tuned, and instruction-tuned). It supports many programming languages, including Python, C++, Java, PHP, Typescript, and C#. Code Llama excels in code completion, debugging, and generating human-readable comments. It features fill-in-the-middle capability and supports large input contexts, making it suitable for complex coding tasks.

Key Features:

- Available in 7B, 13B, and 34B parameter versions.
- Variants include base model, Python fine-tuned, and instruction-tuned.
- Supports many popular programming languages.
- Fill-in-the-middle capability for code completion.
- Large input context support for complex tasks.

Mistral LLM

Mistral LLM, created by Mistral AI, exhibits powerful reasoning abilities, surpassing other top LLM models on benchmarks like HumanEval, MBPP, Math maj@4, GSM8K maj@8 (8-shot), and GSM8K maj@1 (5-shot). It supports function calling and JSON format, making it versatile for various applications.

Key Features:

- Powerful reasoning abilities.
- High performance on benchmarks.
- Supports function calling and JSON format.

Mixtral-8x7B

Mixtral-8x7B, developed by Mistral AI, implements a Mixture of Experts (MoE) architecture, allowing it to achieve high performance in reasoning tasks while being efficient in terms of cost and latency. It outperforms many other models on standard benchmarks.

Key Features:

- Mixture of Experts (MoE) architecture.
- High performance in reasoning tasks.
- Cost and latency efficiency.

Leaderboards to Monitor

Monitoring the performance and biases of LLMs is essential to ensure their effectiveness and fairness. Below are key leaderboards that track these metrics, providing valuable insights into model capabilities and potential biases. These resources help developers choose the right models and make informed deployment decisions, ensuring performance and cultural/social biases are considered.

- **Hugging Face's LLM Performance Leaderboard:** Hugging Face LLM Leaderboard
- **Trustbit LLM Benchmarks:** Trustbit LLM Leaderboard
- **Vectara's Hallucination Leaderboard:** Vectara Hallucination Leaderboard

Cultural/Social Bias Leaderboards

- **Biases LLM Leaderboard by Livable Software:** Livable Software Biases LLM Leaderboard
- **Open LLM Leaderboard by Hugging Face:** Hugging Face Open LLM Leaderboard

7. Anticipating AI-Enabled Hardware

The new line of AI-enabled hardware, such as the upcoming Microsoft Snapdragon laptops set to hit the market sometime in the 3rd quarter of this year, is expected to enter our workplace soon. By adopting the use of AI/LLM locally, we will prepare our Rackers for the proper and secure use of this technology, ensuring they are well-versed in leveraging AI tools effectively and safely from the onset. Microsoft's new feature called Recall, expected in Windows 11, will come preinstalled with the OS, and Apple plans to add local inference to its OS this holiday season.

8. Practical Implementation

To replace traditional AI copilot tools with local LLMs, I utilized an Ubuntu 22.04 workstation. This can also be done using Windows or Mac systems. The process involves:

1. **Using Ollama:** Ollama allows for running LLMs in a Docker container, ensuring isolation from the rest of the machine, enhancing security and compliance.
2. **Running LLM as a Service:** Once the LLM is running as a service, I used the Continue VSCode plugin to connect to my local LLM. This setup allows the LLM to behave in a similar manner to Microsoft's Copilot while ensuring that all code and data remain on my local machine, preventing it from being used by third parties, such as Microsoft, to train their AI models.

This approach demonstrates how we can use LLMs safely and securely, complying with current company policies. By advocating for the company to allocate resources to establish our own trained LLM for internal use, we can significantly enhance productivity and customer support mechanisms.

Commented [TB5]: Maybe we acknowledge the weaknesses in solely relying upon the leaderboards and that models specifically trained in specific areas can lead to greater efficiencies; maybe we suggest that the FAIR team offer recommendations or perhaps specifically trained LLMs or perhaps work with Rackers on the floor to understand our specific needs and offer guidance to other rackers in regards to providing authorized training datasets for rackers to use to train their own LLMs/AI.

Commented [TB6R5]: we could use salted and hashed keys in document meta data identifying the document, what kind of document and if the classification level of that document. AIs/LLM could then be specifically trained on understanding the contents of that data once it has been decrypted by the user decrypting the hash as part of their login creds.

Installing Ollama: NOTE: These instructions assume you have Ubuntu 22.04 and Docker installed on your workstation. It also assumes that you are using NVIDIA hardware (let's add a footnote about why NOT AMD) and that the NVIDIA drivers, including CUDA, are already installed. (Provide a link to more information on how to set that up.) For specific instructions on how to set up Ollama in Windows or Mac, please visit https://ollama.com/ for more information.

1. Open a terminal on your Ubuntu 22.04 workstation.
2. Pull the Ollama Docker image by running the command docker pull ollama/ollama.
3. Create a new directory for your Ollama configuration files, e.g., mkdir ~/ollama-config.

Configuring Ollama:

1. Create a config.json file in the ~/ollama-config directory with the following content:

```
{
  "storage": {
    "path": "/path/to/storage"
  },
  "api": {
    "port": 8080
  }
}
```

NOTE: Replace /path/to/storage with the actual path where you want to store documents and data that the LLM will access. Ensure the LLM operates under the principle of 'least privilege' by restricting its access only to this designated storage path and not the entire filesystem. This helps maintain data integrity and context for the LLM.

Running Ollama as a Service:

1. Create a docker-compose.yml file in the ~/ollama-config directory with the following content:

```
version: '3'
services:
  ollama:
    image: ollama/ollama
    command: ollama run <model-name>
    volumes:
      - ./config.json:/app/config.json
      - /path/to/storage:/app/storage
    ports:
      - "8080:8080"
    deploy:
      resources:
        reservations:
          devices:
            - capabilities: [gpu]
    runtime: nvidia
```

Replac models page. Ensure /path/to/storage matches the path specified in config.json.

1. Run the command docker-compose up -d to start Ollama as a service in detached mode.

Connecting to Ollama API:

1. Open VSCode and install the "Continue" plugin.
2. Configure the plugin to connect to your local Ollama API by setting the ollama.url parameter to http://localhost:8080.

This setup ensures that each container can run a different model, promoting modularity and ease of management. By following these steps, you can use your local LLM as a copilot, ensuring that all code and data remain on your local machine, complying with company policies.

9. Conclusion

Adopting a secure and compliant approach to local LLM deployment allows us to unlock the full potential of AI-driven productivity while maintaining our organization's commitment to protecting sensitive information. By prioritizing security considerations, complying with organizational policies, and utilizing open-source software for model updates and maintenance, we can enhance productivity, improve security measures, and comply with existing policies while driving innovation and growth within our organization.